The Analysis Research of Hierarchical Storage System Based on Hadoop Framework

Yan LIU[1, a], Tianjian ZHENG[1], Mingjiang LI[1], Jinpeng YUAN[1]

[1] Digital engineering research center for Area national culture,Qiannan Normal College For Nationgalities, DuYun 558000,China

[a] Liuyan300@sina.com

Keywords: Big Data, Hierarchical Storage System, Hadoop Framework.

Abstract. With the rapid development of data science and computer technology, hierarchical storage system based on Hadoop is being research. In this paper, we propose a novel methodology of hierarchical storage system based on Hadoop framework. This way the Hadoop cluster will not reach the stage where the NameNode becomes irresponsive due to excessive JVM garbage collection as the HDFS will not be heavily loaded. The experimental result illustrates the effectiveness and feasibility of proposed framework, further modification areas of research are proposed in the end.

Introduction

The Hadoop Distributed File System (HDFS) is designed as a massive data storage framework and serves as the storage component for the Apache Hadoop platform. The file system is based on hardware and provides a highly reliable storage and high throughput of global access to large data sets. Due to these advantages, HDFS also as an independent general distributed file system and non-Hadoop services application [1-3]. However, the advantage of HDFS provides high throughput degradation rapidly when handling interaction-intensive files, namely small file size, but often visit. Reason is that before the start of an I/O transmission, there are some necessary initialization steps need to be done, etc. data location retrieving and storage space allocation. When transferring large data, this initialization overhead becomes relatively small and can be negligible compared with the data transmission itself. However, when transferring small size data, this overhead becomes significant. In addition to the initialization overhead, files with high I/O data access frequencies can also quickly overburden the regulating component in the HDFS, i.e., the single name-node that supervises and manages every access to data-nodes [4]. If the number of data-nodes is large, the single name-node can quickly become a bottleneck when the frequency of I/O requests is high.

Hadoop was first developed as a Big Data processing system in 2006 at Yahoo! The idea is based on Google's Map-Reduce, which was first published by Google based on their proprietary Map-Reduce implementation. In the past few years, Hadoop has become a widely used platform and runtime environment for the deployment of Big Data applications. Hadoop core including the file system, processing, and computing resources, also provides the basic services to build a cloud computing environment and commercial software. Hadoop provides a distributed file system and a framework for analysis and transformation of very large data sets using Hadoop computing component graphs paradigm. Hadoop cluster scale of computing power, storage capacity and I/O bandwidth by simply increasing commodity servers. The Hadoop distributed file system (HDFS) is Hadoop file system component data is stored in which place [5]. HDFS master/slave architecture. An HDFS cluster consists of a single Name-nodes, a master server management file system namespace and adjust the customer access to the file. In addition, there are a number of data-nodes, usually each node in a cluster, manage storage run on nodes. Name-Node constantly refer to each file and the file system block in memory, this means that in the cluster, with many files are very big limiting factor for the extended memory so when it is difficult for a memory is full of the Name-Node just add another cluster nodes for further storage needs the Name-Node will not be able to handle these additional nodes and file system metadata will be generated [6-7]. In many systems,

© 2015. The authors - Published by Atlantis Press

frequent file access is unavoidable. For example, log file updating, which is a common procedure in many applications. Since the HDFS applies the rule of ‘‘Write-Once–Read-Many’’, the updating procedure first reads these files, modifies them and then writes them back. Such an updating procedure generates I/O requests with high frequency. Another example is the synchronization procedure in a distributed system using incremental message delivery via the file system. In this case, an incremental message file is generated by a changed component of a distributed system. This file is relatively small and it is read by all participating components to synchronize the entire system. As the HDFS allows multiple tasks reading the same file simultaneously, it is possible that a file is read frequently within a short period of time [8-10].

In this paper, we use the HDFS as a stand-alone file system and present an integrated approach to addressing the HDFS performance degradation issue for interaction-intensive tasks. We will discuss the issue in detail in the next sections.

Related Research and Literature Review

The ever growing technology has resulted in the need for storing and processing excessively large amounts of data. The current volume of data is enormous and is expected to replicate over 650 times by end the year 2014, out of which, 85% would be unstructured. This is known as the Big Data problem. Apache Hadoop is an open source project construction and operation of the Apache software foundation is the international community of contributors and users. Hadoop is a distributed batch Hadoop infrastructure currently used in big data management is designed to be parallel and elasticity. It redefines the data management and processing way of using the hardware composition of the power of computing resources.

Architecture of Hadoop System. Hadoop Distributed File System is a fault-tolerant distributed file system designed to run on "off-the-shelf" hardware. It has been optimized for streaming reads on large files whereas I/O throughput is favored over low latency. In addition, HDFS uses a simple model for data consistency where files can only be written to once. HDFS hypothesis disk failure may be the case, and use a concept called copy duplicate data block in the nodes in the cluster. HDFS using a larger block size than desktop file system. HDFS, for example, the default block size is 64 MB. Once in a HDFS file, the file is divided into one or more data blocks and distributed to the nodes in the cluster. In addition, a copy of the data and redistribute the cluster nodes to ensure high availability data to disk failure.

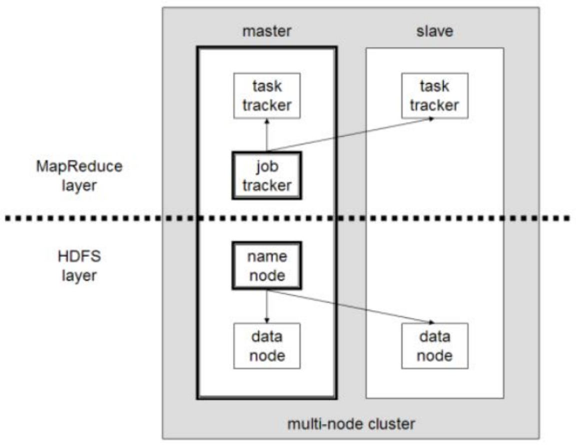

Figure 1: The Hadoop Architecture

DataNodes are the workers of the file system. DataNode performs creation, deletion and copy of block under the NameNode's command. They store and retrieve blocks when they are told to (by clients or the NameNode), and they report back to the NameNode periodically with lists of blocks that they are storing. Each block replica on a DataNode is represented by two files in the local hosts native file system. MapReduce is a programming model for data processing. Hadoop can run MapReduce programs written in various languages. A MapReduce program is composed of a Map() procedure that performs filtering and a Reduce() procedure that performs a summary operation [11].

MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer. The programmer also specifies two functions: the map function and the reduce function. Working here application is divided into many small pieces, each of which can be executed or to perform any node in the cluster. Hadoop is an important feature of partition of data and calculation in many thousands of the host, and execute the application parallel computing examples close to their data through the use of graphs. In the nutshell in a typical MapReduce job, multiple map tasks on slave nodes are executed in parallel, generating results buffered on local machines. Once some or all of the map tasks have finished, the shuffle process begins, which aggregates the map task outputs by sorting and combining key-value pairs based on keys. Then, the shuffled data partitions are copied to reducer machine.

Hadoop and Hadoop Ecosystem. Although Hadoop is best known for MapReduce and its distributed file system (HDFS, renamed from NDFS), the term is also used for a family of related projects that fall under the umbrella of infrastructure for distributed computing and large-scale data processing.

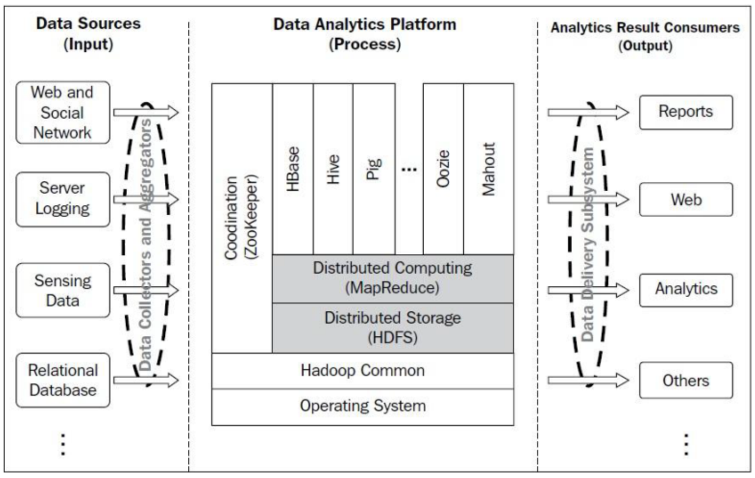

Figure 2: The Hadoop Ecosystem

Drawbacks of the Current Systems. The NameNode server in the Hadoop cluster keeps the track of filesystem meta-data, it keeps a track of how your files are broken down into file blocks, which nodes store those blocks, and the overall health of the distributed filesystem. From the NameNode is a single container file system metadata, it naturally became the file system constraints. In order to make the metadata operations is fast, the NameNode the namespace loaded into memory, so the size of the namespace is a limited number of available RAM the NameNode. Therefore, the available memory NameNode is no chance to limit the number of customer cluster growth. Is also a big HDFS install one operation in a large memory space of the NameNode JVM is susceptible to almost all frequent garbage collection, this may require the NameNode service for a few minutes. Thus it is clear that memory available to the NameNode machine in a Hadoop cluster dictates the size of cluster and ultimately the number of active clients using a Hadoop based application.

The Proposed Methodology

Extended Namenode Structure. The original HDFS structure, which consists of a single namenode, is redesigned with a two-layer namenode structure as shown in Fig. 3. The first layer contains a Master Name Node (MNN) which is actually the namenode in the original HDFS structure. The second layer consists of a set of Secondary Name Nodes (SNNs), which are applied to every rack of datanodes.

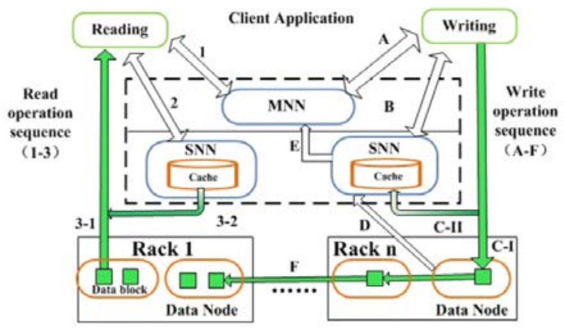

Figure 3: The Hierarchical Namenode Structure

Similarly, to write data into a datenote, the MNN first allocates one or more racks for the client application to write (Step A). Then, the SNN further allocates datanodes for the client application (Step B). Afterwards, the client application begins to write file blocks into these datanodes.

Overview of the Design Pattern. The basic concept of cache i.e. storing the frequently used data closer to the processor than the whole data and using it for faster operation can be applied in the NameNode memory management. Thus when the NameNode is operating in a heavy HDFS installation, the metadata records of frequently used data that is being used by most of the active clients can be kept in the NameNode memory space and the metadata records of least used data which sometimes can even be irrelevant can be kept at a different location and accessed as and when required. Such a large amount of memory used to reduce the NameNode and then further data can be used to store more frequently. There are less loaded into the JVM garbage collection of HDFS NameNode will reduce this will avoid the NameNode become unresponsive problem due to excessive garbage collection.

The Model Architecture

In this architecture, along with all the basic block of Hadoop a hardware connection. This increases the hardware is quite rapid access storage device. It as a least recently used secondary storage devices to store metadata. Metadata is stored in it will refer to only when the request for access to the data is used less often.

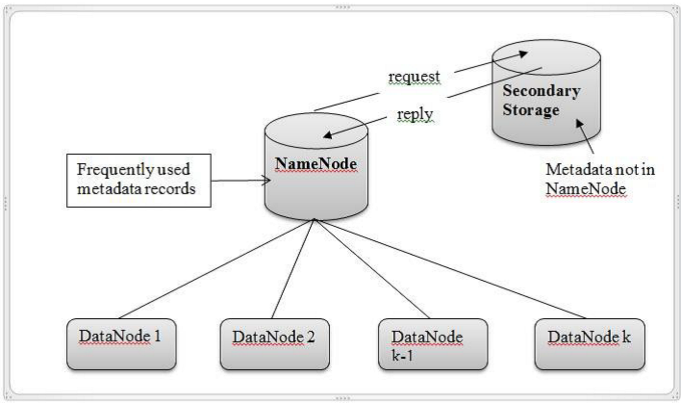

Figure 4: The Hierarchical Structure

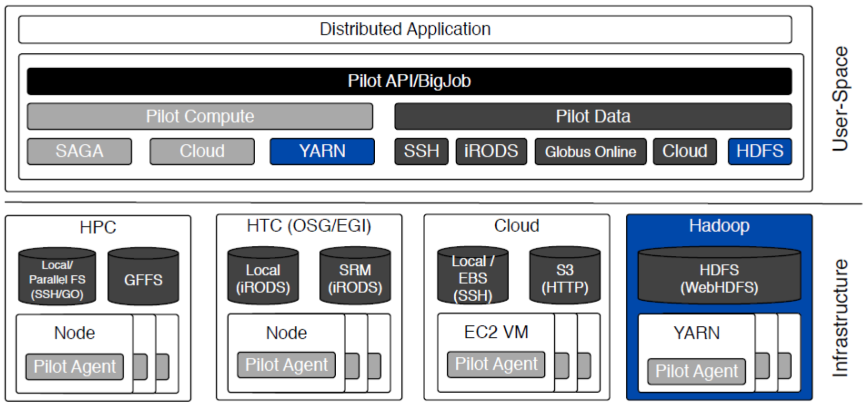

Figure 5: The Advanced Architecture

When the client performs a read operation the usual processing that is done by the NameNode to fetch metadata for requested file will be done and the client will be given back the file. The only difference will occur when the requested file is moved to the secondary storage as it was not recently used and the NameNode has reached threshold value. In this case the NameNode will not find the record in its memory and thus a request will be made to secondary device to fetch the metadata record and the metadata record will be removed from secondary device and will be loaded again on the RAM.

The Model Implementation. As some of the files were left untouched and other were accessed randomly, the separation module remove approx. 0.3 Million files the first time and 0.25 million files for the second time and 0.28 million for the third time. This removal of the files has made available approximately 250 MB of RAM. Due to this free space, the Hadoop does not become unresponsive. The free space aids in faster operation by Hadoop NameNode as well as by local operating system.

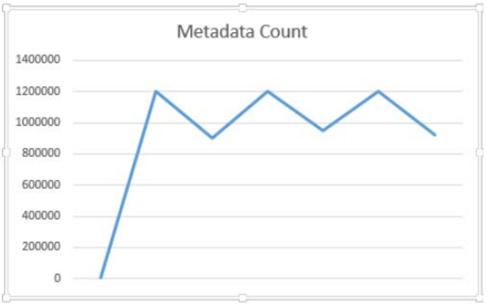

Figure 6: The Experiment Result

An observation which can trigger is that every time our separation algorithm of data in main memory still NameNode in fact metadata recently added or often use and access. Read operations in the future it can be seen as a positive point of view, because whenever I read requests search operation for the NameNode returns the latest data.

Conclusions

In this paper, we propose a novel methodology of hierarchical storage system based on Hadoop framework. This way the Hadoop cluster will not reach the stage where the NameNode becomes irresponsive due to excessive JVM garbage collection as the HDFS will not be heavily loaded. Also as the NameNode will only store relatively more frequently used data the operations carried on the cluster will be faster and more efficient. Compared with our active switch out approach, it may have potential advantages over average cases. We will further investigate other newly emerged solutions. In particular, without considering that it may need client side code modification, the HDFS structure provided in previous section seems to have advantages over our two-layer structure and is worth further investigation and comparison.

References

[1] Mukhopadhyay, Debajyoti, et al. "Addressing NameNode Scalability Issue in Hadoop Distributed File System using Cache Approach." arXiv preprint arXiv:1411.6775 (2014).

[2] Le, Hieu Hanh, Satoshi Hikida, and Haruo Yokota. "NDCouplingHDFS: A Coupling Architecture for a Power-Proportional Hadoop Distributed File System." IEICE TRANSACTIONS on Information and Systems 97.2 (2014): 213-222.

[3] Le, Hieu Hanh, Satoshi Hikida, and Haruo Yokota. "NDCouplingHDFS: A Coupling Architecture for a Power-Proportional Hadoop Distributed File System." IEICE TRANSACTIONS on Information and Systems 97.2 (2014): 213-222.

[4] Kim, Sang Don, et al. "Compression Accelerator for Hadoop Appliance." Internet of Vehicles–Technologies and Services. Springer International Publishing, 2014. 416-423.

[5] Dwivedi, Kalpana, and Sanjay Kumar Dubey. "Analytical review on Hadoop Distributed file system." Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference-. IEEE, 2014.

[6] Zhao, Tiezhu, Zusheng Zhang, and Huaqiang Yuan. "Analysis of Distributed File Systems on Virtualized Cloud Computing Environment." Computer Engineering and Networking. Springer International Publishing, 2014. 817-823.

[7] Chang, Ruay-Shiung, et al. "Dynamic Deduplication Decision in a Hadoop Distributed File System." International Journal of Distributed Sensor Networks 2014 (2014).

[8] Gupta, Himanshu, et al. "Systems and methods for massive structured data management over cloud aware distributed file system." U.S. Patent No. 8,775,425. 8 Jul. 2014.

[9] Soundarabai, P. Beaulah, K. R. Venugopal, and L. M. Patnaik. "ISSN: 2277-3754 ISO 9001: 2008 Certified Big Data Analytics: An Approach using Hadoop Distributed File System." (2014).

[10] Zhang, Tao, et al. "ParSA: High-throughput scientific data analysis framework with distributed file system." Future Generation Computer Systems (2014).

[11] Zhao, Tiezhu, Xin Ao, and Huaqiang Yuan. "Modeling and Evaluation of the Performance of Parallel/Distributed File System." Computer Engineering and Networking. Springer International Publishing, 2014. 421-427.